



## Recurrent Spatial Transformer Networks

**Sønderby, Søren Kaae; Sønderby, Casper Kaae; Maaløe, Lars; Winther, Ole**

*Published in:*  
arXiv

*Publication date:*  
2015

*Document Version*  
Publisher's PDF, also known as Version of record

[Link back to DTU Orbit](#)

*Citation (APA):*  
Sønderby, S. K., Sønderby, C. K., Maaløe, L., & Winther, O. (2015). Recurrent Spatial Transformer Networks. *arXiv*, [arXiv:1509.05329].

---

### General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

---

# Recurrent Spatial Transformer Networks

---

Søren Kaae Sønderby<sup>1</sup>  
Casper Kaae Sønderby<sup>1</sup>  
Lars Maaløe<sup>2</sup>  
Ole Winther<sup>1,2</sup>

SKAAESONDERBY@GMAIL.COM  
CASPERKAAE@GMAIL.COM  
LARSMA@DTU.DK  
OLWI@DTU.DK

<sup>1</sup> Bioinformatics Centre, Department of Biology, University of Copenhagen, Copenhagen, Denmark

<sup>2</sup> Department for Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Lyngby, Denmark

## Abstract

We integrate the recently proposed spatial transformer network (SPN) (Jaderberg & Simonyan, 2015) into a recurrent neural network (RNN) to form an RNN-SPN model. We use the RNN-SPN to classify digits in cluttered MNIST sequences. The proposed model achieves a single digit error of 1.5% compared to 2.9% for a convolutional networks and 2.0% for convolutional networks with SPN layers. The SPN outputs a zoomed, rotated and skewed version of the input image. We investigate different down-sampling factors (ratio of pixel in input and output) for the SPN and show that the RNN-SPN model is able to down-sample the input images without deteriorating performance. The down-sampling in RNN-SPN can be thought of as adaptive down-sampling that minimizes the information loss in the regions of interest. We attribute the superior performance of the RNN-SPN to the fact that it can attend to a sequence of regions of interest.

convolutional network is used to predict a sequence of digits. Because the predictions are made with  $n$  independent softmax layers at the top of the network, the SPN layers must find the box containing the entire sequence. A better model would sequentially produce the targets by locating and zoom in on each individual element before each prediction. In this paper we combine the SPN network with a recurrent neural network (RNN) to create a recurrent SPN (RNN-SPN) that sequentially zooms in on each element. This is illustrated in Figure 1 panel b). In the SPN-RNN a RNN network produces inputs for the SPN at each time-step. Using these inputs the SPN produces a transformed version of the input image which is then used for classification. By running the recursion for multiple steps and conditioning each transformation on the previous time-step the RNN-SPN model can sequentially attend to the part of the image containing each elements of interest and only use the relevant information for classification. Because these regions are generally small we experiment with forcing the RNN-SPN to down-sample the image which can thought of as adaptive down-sampling thus keeps the resolution of the regions of interest (nearly) constant.

## 1. Introduction

Attention mechanisms have been used for machine translation (Bahdanau et al., 2014), speech recognition (Chorowski et al., 2014) and image recognition (Ba et al., 2014; Gregor et al., 2015; Xu et al., 2015). The recently proposed spatial transformer network (SPN) (Jaderberg & Simonyan, 2015) is a new method for incorporating spatial attention in neural networks. SPN uses a learned affine transformation of the input and bilinear interpolation to produce its output. This allows the SPN network to zoom, rotate and skew the input. A SPN layer can be used as any other layer in a feed-forward convolutional network<sup>1</sup>. The feed-forward SPN (FFN-SPN) is illustrated in Figure 1 panel a) where a SPN combined with a

## 2. Related Work

Gregor et al. 2015 introduced a differentiable attention mechanism based on an array of Gaussians and combined it with a RNN for both generative and discriminative tasks. Ba et al. 2014 and Sermanet et al. combined a non-differentiable attention mechanism with an RNN and used it for classification. Their attention mechanism was trained using reinforcement learning. Other related work include (Xu et al., 2015) who applies visual attention in an encoder-decoder structure.

## 3. Spatial Transformer Network

The SPN network is implemented similarly to (Jaderberg & Simonyan, 2015). A SPN network takes an

---

<sup>1</sup>See e.g. <https://goo.gl/1Ho1M6>

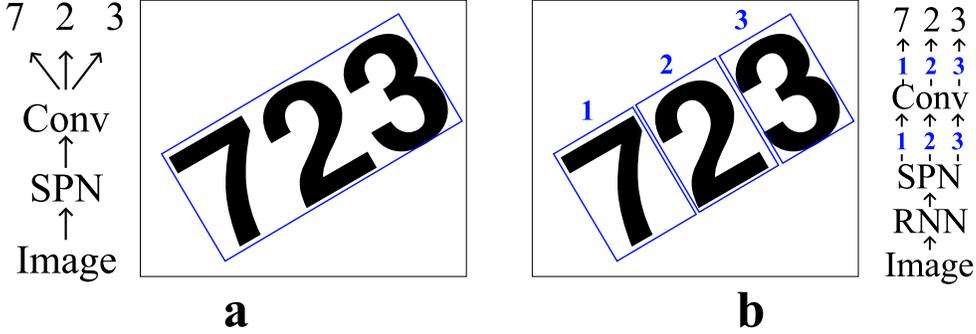


Figure 1. SPN networks for predicting the sequence 723 from an image. a) A FFN-SPN network attend to the entire sequence (blue box). The digits are classified by three separate softmax layers at the top of the network. Because the network cannot zoom in on individual digits in the sequence all digits are classified from the same image crop. b) A RNN-SPN where the transformation is predicted with an RNN. This allows the model to create a separate crop for each digit. Each crop (indicated with blue numbers) is then passed through the same classification network. The structure enables the model to zoom in on each individual digit.

image or a feature map from a convolutional network as input. An affine transformation and bilinear interpolation is then applied to the input to produce the output of the SPN. The affine transformation allows zoom, rotation and skew of the input. The parameters of the transformation are predicted using a localization network  $f_{loc}$ :

$$f_{loc}(I) = \mathbf{A}_\theta = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \end{bmatrix}, \quad (1)$$

where  $I$  is the input to the SPN with shape  $[H \times W \times C]$  (height, width, channels) and matrix  $\mathbf{A}_\theta$  specifies the affine transformation. The affine transformation is applied on a mesh grid  $\mathbf{G} \in \mathcal{R}^{h \times w}$ :

$$\mathbf{G} = \{(y_1, x_1), (y_1, x_2), \dots, (y_2, x_1), (y_2, x_2), \dots, (y_w, x_{h-1}), (y_h, x_w)\}. \quad (2)$$

where  $w$  and  $h$  does not need to be equal to  $H$  and  $W$ .  $\mathbf{G}$  is illustrated in Figure 2 panel a). The grid is laid out such that  $(y_1, x_1) = (-1, -1)$  and  $(y_h, x_w) = (+1, +1)$  with the points in-between spaced equally. The affine transformation is applied to  $\mathbf{G}$  to produce an image  $\mathbf{S}$  which tells how to select points from  $I$  and map them back onto  $\mathbf{G}$ :

$$\mathbf{S}_{ij} = \mathbf{A}_\theta \begin{bmatrix} y_i \\ x_j \\ 1 \end{bmatrix} \quad (3)$$

where we have augmented each point with a 1. Since the mapped points in  $\mathbf{S}$  does not correspond exactly to one pixel in  $I$  bilinear interpolation is used to interpolate each point in  $\mathbf{S}$ . The sub-gradients for the bilinear interpolation are defined and we can use standard backpropagation through the transformation to learn the parameters in  $f_{loc}$ . The sampling process is illustrated in Figure 2 where panel a) illustrates  $\mathbf{G}$  and panel b) illustrates  $\mathbf{S}$  after we have applied the transformation.

### 3.1. Down-sampling

We can vary the number of sampled points by varying  $h$  and  $w$ . Having  $h$  and  $w$  less than  $H$  and  $W$  will downsample the input to the SPN. We specify the down sampling with  $d$ .  $d$  larger than 1 will downsample the input. The number of sampled points form  $I$  is:

$$n_{points} = \left(\frac{H}{d}\right) \cdot \left(\frac{W}{d}\right) = \frac{H \cdot W}{d^2}. \quad (4)$$

For 2D images the sampled points decrease quadratically with  $d$ .

### 3.2. RNN-SPN

In the original FFN-SPN the localization network is a feed-forward convolutional neural network. We modify this model by letting an RNN predict the transformation matrices such that

$$c = f_{conv}(I) \quad (5)$$

$$h_t = f_{loc}^{rnn}(c, h_{t-1}) \quad (6)$$

$$\mathbf{A}_\theta = g(h_t). \quad (7)$$

where  $f_{conv}$  is a convolutional network taking  $I$  as input and creating a feature map  $c$ ,  $f_{loc}^{rnn}$  is an RNN, and  $g$  is a FFN. Here an affine transformation is produced at each time-step from the hidden state of the RNN. Importantly the affine transformations are conditioned on the previous transformations through the time dependency of the RNN.

## 4. Experiments

We test the model on a dataset of sequences of MNIST digits cluttered with noise. The dataset was created by placing 3 random MNIST digits on a canvas of size  $100 \times 100$  pixels. The first digits was placed by randomly sampling an

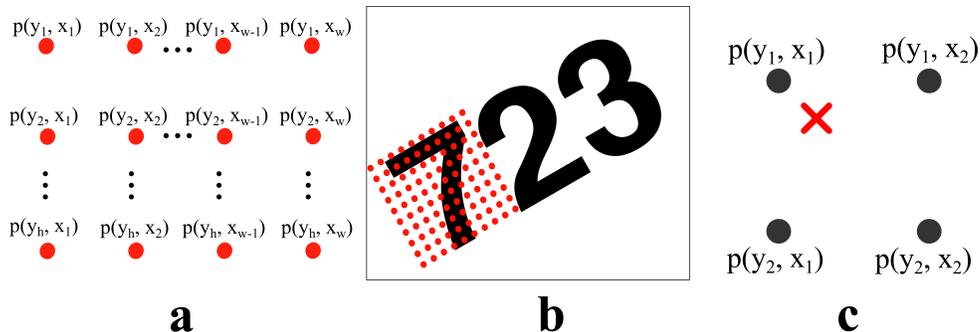


Figure 2. a) The sampling grid  $\mathbf{G}$  of equally spaced sampling points. We set  $y_1$  and  $x_1$  to  $-1$  and  $y_h, x_w$  to  $+1$ . b) A recurrent SPN is able to zoom in on each element in the sequence. c) Bilinear transformation will interpolate the red cross by calculating a weighted average of the four nearest pixels. The operation is differentiable.

$y$  position on the canvas. The  $x$  positions were randomly sampled subject to the entire sequence must fit the canvas and the digits are non-overlapping. Subsequent digits are placed by following a slope sampled from  $\pm 45^\circ$ . Finally the images are cluttered by randomly placing 8 patches of size  $9 \times 9$  pixels sampled from the original MNIST digits. For the test, validation and training sets we sample from the corresponding set in the original MNIST dataset. We create 60000 examples for training, 10000 for validation, and 10000 for testing<sup>2</sup>. Figure 3 shows examples of the generated sequences.

As a baseline model we trained a FNN-SPN with the SPN layer following immediately after the input. The classification network had 4 layers of conv-maxpool-dropout layers followed by a fully connected layer with 400 units and finally a separate softmax layer for each position in the sequence. The convolutional layers had 96 filters with size  $3 \times 3$  and rectified linear units were used for nonlinearity in both the convolutional and fully connected layers. For comparison we further train a purely convolutional network similar to the classification network used in the FNN-SPN.

The RNN-SPN use a gated recurrent unit (GRU) (Chung et al., 2014) with 256 units. The GRU is run for 3 time steps. At each time step the GRU unit use  $c$  as input. We apply a linear layer to convert  $h_t$  into  $\mathbf{A}_\theta^t$ . The RNN-SPN is followed by a classification convolutional network similar to the network used in the FNN-SPN model, except that the convolutional layers only have 32 filters.

In all experiments, the localization networks had 3 layers of max-pooling convolutional layers. All convolutional layers had 20 filters with size  $[3 \times 3]$ . All models were trained with RMSprop (Tieleman & Hinton, 2012) down-sampling factors and dropout rates optimized on the validation set. A complete description of the models can be found in the

<sup>2</sup>The script for generating the dataset is available along with the rest of the code.

Table 1. Per digit error rates on MNIST sequence dataset,  $d$  is the down-sampling factor.

Cluttered MNIST Sequences	
Model	Err. (%)
RNN-SPN $d=1$	1.8
RNN-SPN $d=2$	<b>1.5</b>
RNN-SPN $d=3$	1.8
RNN-SPN $d=4$	2.3
FFN-SPN $d=1$	4.4
FFN-SPN $d=2$	2.0
FFN-SPN $d=3$	2.9
FFN-SPN $d=4$	5.3
Conv. net.	2.9

Appendix.

The models were implemented using Theano (Bastien et al., 2012) and Lasagne (Dieleman et al., 2015). The SPN has been merged into the Lasagne library<sup>3</sup>. Code for models and dataset is released at <https://goo.gl/RspkZy>.

## 5. Results

Table 1 reports the per digit error rates for the tested models. The RNN-SPN models perform better than both convolutional networks (2.9%) and FFN-SPN networks (2.0%). In Figure 3 we show where the model attend on three sample sequences from the test set. The last three columns show image crops after the affine transformation using a down-sampling factor of three. We found that increasing the down-sampling factor above one encouraged the model to zoom. When the down-sampling factor is greater than one we introduce an information bottleneck forcing the model to zoom in on each digit. The poor performance of the FFN-

<sup>3</sup>Available here: <http://goo.gl/kgSk0t>

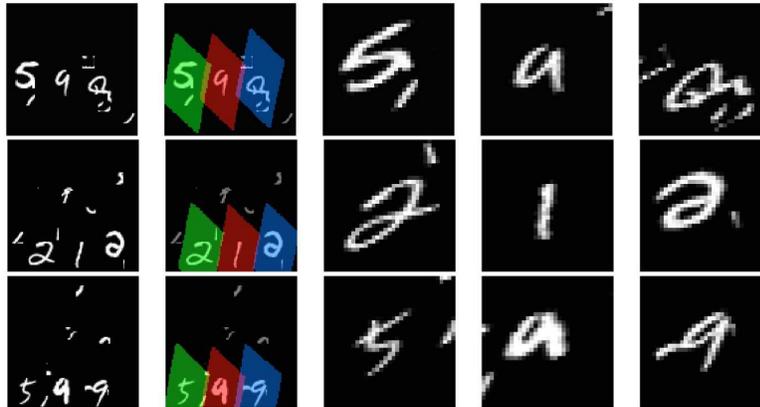


Figure 3. The left column shows three examples of the generated cluttered MNIST sequences. The next column shows where the model attend when classifying each digit. In the last 3 column we show the image crops that the RNN-SPN uses to classify each digit. The input sequences are  $100 \times 100$  pixels. Each image crop is  $33 \times 33$  pixels because the model uses a down-sample factor of 3.

SPN convolutional net for high down-sampling values is explained by the effective decrease in resolution since the model needs to fit all three digits in the image crop.

## 6. Conclusion

We have shown that the SPN can be combined with an RNN to classify sequences. Combining RNN and SPN creates a model that performs better than FFN-SPN for classifying sequences. The RNN-SPN model is able to attend to each individual element in a sequence, something that the FFN-SPN network cannot do. The main advantage of the RNN-SPN model when compared to the DRAW network (Gregor et al., 2015) is that the SPN attention is faster to train. Compared with the model of (Mnih et al., 2014) our model is end-to-end trainable with backpropagation. In this work we have implemented a simple RNN-SPN model future work include allowing multiple glimpses per digit and using the current glimpse as input to the RNN network.

## References

- Ba, Jimmy, Mnih, Volodymyr, and Kavukcuoglu, Koray. Multiple Object Recognition with Visual Attention. December 2014.
- Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. September 2014.
- Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian, Bergeron, Arnaud, Bouchard, Nicolas, Warde-Farley, David, and Bengio, Yoshua. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, November 2012.
- Chorowski, Jan, Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. End-to-end Continuous Speech Recognition using Attention-based Recurrent NN: First Results. December 2014.
- Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv:1412.3555*, December 2014.
- Dieleman, Sander, Schlüter, Jan, Raffel, Colin, Olson, Eben, Sønderby, Søren Kaae, Nouri, Daniel, Battenberg, Eric, van den Oord, Aaron, and other contributors. Lasagne: First release., August 2015. URL <http://dx.doi.org/10.5281/zenodo.27878>.
- Gregor, Karol, Danihelka, Ivo, Graves, Alex, and Wierstra, Daan. DRAW: A Recurrent Neural Network For Image Generation. *arXiv preprint arXiv:1502.04623*, 2015.
- Jaderberg, M and Simonyan, K. Spatial Transformer Networks. *arXiv preprint arXiv: 1506.02025*, 2015.
- Mnih, Volodymyr, Heess, Nicolas, Graves, Alex, and Kavukcuoglu, Koray. Recurrent Models of Visual Attention. June 2014.
- Sermanet, Pierre, Frome, Andrea, and Real, Esteban. Attention for Fine-Grained Categorization.
- Tieleman, T and Hinton, Geoffrey E. {Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude}, 2012.
- Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Cho, Kyunghyun, Courville, Aaron, Salakhutdinov, Ruslan, Zemel, Richard, and Bengio, Yoshua. Show, Attend and Tell:

Neural Image Caption Generation with Visual Attention.  
February 2015.

---

# Appendix

---

## 6.1. RNN-SPN

### Localisation network

1. maxpool(2,2)
2. conv(20@(3,3))
3. maxpool(2,2)
4. conv(20@(3,3))
5. maxpool(2,2)
6. conv(20@(3,3))
7. GRU(units=256)
8. Denselayer(6, linear output)

### Classification network

1. SPATIAL TRANSFORMER LAYER (downsample = 1, 3, 4)
2. conv(32@(3,3))
3. maxpool(2,2)
4. dropout(p)
5. conv(32@(3,3))
6. maxpool(2,2)
7. dropout(p)
8. conv(32@(3,3))
9. dropout(p)
10. Dense(256)
11. softmax

In table Table 1 the model are reported at RNN SPN with an entry for each tested downsample factor.

## 6.2. Baseline models

### FFN-SPN model

1. SPATIAL TRANSFORMER LAYER (downsample = 2.0)
2. conv(96@(3,3))
3. maxpool(2,2)
4. dropout(p)
5. conv(96@(3,3))
6. maxpool(2,2)
7. dropout(p)
8. conv(96@(3,3))
9. dropout(p)
10. Dense(400)
11.  $3 \times$  softmax

### Localisation network

1. maxpool(2,2)
2. conv(20@(3,3))
3. maxpool(2,2)
4. conv(20@(3,3))
5. maxpool(2,2)
6. conv(20@(3,3))
7. Denselayer(200)
8. Denselayer(6, linear output)

This model is reported in Table 1 as the SPN conv. network. Finally we also trained the classification network from the FFN-SPN reported as conv. Net in Table 1.